TRABAJO FIN DE GRADO

TÍTOL DEL TFG: Visualización objetos 3D en mapas marítimos

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Sergey Volokhov

DIRECTOR: Dolors Royo

DATA: 8 de febrero del 2017

Títol: Visualización objetos 3D en mapas marítimos

Autor: Sergey Volokhov

Director: Dolors Royo

Data: Febrero 8 2017

Resumen

El proyecto Pilotfish forma parte del proyecto Artifact que en un futuro debería ofrecer la posibilidad de visualizar los datos en una plataforma HMD (Head Mounted Device). El objetivo es ampliar la seguridad en la navegación permitiendo la visualización de la información necesaria en el campo de visión.

El objetivo de este proyecto es ampliar los datos procesados en la aplicación Pilotfish desarrollada por el “Department of Computer Architecture (EETAC)”. Que inicialmente procesa y representa los datos de posición y profundidad obtenidos de los sensores de un barco. Este proyecto añade al Pilotfish la posibilidad de procesar y representar los datos de velocidad y heading del barco, velocidad y dirección del viento y la temperatura del agua.

Las tecnologías implicadas en el proyecto son el estándar NMEA 0183, en este formato se guarda la información recibida por los sensores del barco. El servidor SignalK que se dedica a procesar las sentencia NMEA 0183 y guardar las en el objeto SignalK que tiene formato json y es fácil de interpretar posteriormente. Después se utiliza el cliente SignalK que se dedica a procesar el objeto json y mostrar los datos que lleva por la pantalla para su interpretación visual.

En el trabajo se ha utilizado un servidor SignalK desarrollado por la UPC con el objetivo de añadir los parsers que permitan la traducción de las sentencias necesarias de NMEA0183 a los objetos SignalK.

El servidor recibe las sentencias NMEA del fichero generado por una embarcación, los parsers recogen información de las sentencias y las asignan a un objeto SignalK. Luego con el cliente se puede extraer esta información suscribiéndose a los datos necesarios (profundidad, velocidad, heading) para hacer la lectura y visualizar estos datos.

Title: Visualización objetos 3D en mapas marítimos

Author: Sergey Volokhov

Director: Dolors Royo

Date: Febrero 8 2017

Resumen

El proyecto Pilotfish forma parte del proyecto Artifact que en un futuro debería ofrecer la posibilidad de visualizar los datos en una plataforma HMD (Head Mounted Device). El objetivo es ampliar la seguridad en la navegación permitiendo la visualización de la información necesaria en el campo de visión.

El objetivo de este proyecto es ampliar los datos procesados en la aplicación Pilotfish desarrollada por el “Department of Computer Architecture (EETAC)”. Que inicialmente procesa y representa los datos de posición y profundidad obtenidos de los sensores de un barco. Este proyecto añade al Pilotfish la posibilidad de procesar y representar los datos de velocidad y heading del barco, velocidad y dirección del viento y la temperatura del agua.

Las tecnologías implicadas en el proyecto son el estándar NMEA 0183, en este formato se guarda la información recibida por los sensores del barco. El servidor SignalK que se dedica a procesar las sentencia NMEA 0183 y guardar las en el objeto SignalK que tiene formato json y es fácil de interpretar posteriormente. Después se utiliza el cliente SignalK que se dedica a procesar el objeto json y mostrar los datos que lleva por la pantalla para su interpretación visual.

En el trabajo se ha utilizado un servidor SignalK desarrollado por la UPC con el objetivo de añadir los parsers que permitan la traducción de las sentencias necesarias de NMEA0183 a los objetos SignalK.

El servidor recibe las sentencias NMEA del fichero generado por una embarcación, los parsers recogen información de las sentencias y las asignan a un objeto SignalK. Luego con el cliente se puede extraer esta información suscribiéndose a los datos necesarios (profundidad, velocidad, heading) para hacer la lectura y visualizar estos datos.

ÍNDICE

INTRODUCCIÓN

En este documento se describe la ampliación del proyecto Pilotfish que en su tiempo es una primera etapa del proyecto ARtifact. El proyecto realizado es de visualización de datos en el mapa marítimo, en concreto en el mapa del cliente del proyecto Pilotfish recogido desde el servidor SignalK a través de una suscripción. El servidor en su tiempo recibe la información a través de los mensajes NMEA0183 que provienen de los sensores del barco.

El documento está dividido en los capítulos siguientes:

En el **capítulo 1** se describe la jerarquía de los proyectos y sus objetivos, se explica el proyecto ARtifact, se habla del proyecto Pilotfish como una etapa inicial de este proyecto.

En el **capítulo 2** se habla de las mejoras introducidas por el proyecto de visualización de datos en la etapa del parser y manipulación de las sentencias NMEA0183 y su conversión en el objeto SignalK.

En el **capítulo 3** se introduce el formato SignalK y se habla de sus ventajas e inconvenientes y porque se ha utilizado en el proyecto.

En el **capítulo 4** después se habla sobre el mismo servidor SignalK, su estructura y funcionamiento.

Y finalmente en el **capítulo 5** se habla de la etapa final del proyecto donde se visualizan los datos que es el cliente SignalK.

Durante el proyecto se han utilizado y manipulado el servidor SignalK de Pilotfish en el lenguaje C#, se ha estudiado el formato NMEA0183 y se ha implementado su conversión al formato SignalK, y finalmente se ha utilizado el Unity3D para la parte del trabajo con el cliente.

CAPÍTULO 1. Visualización de objetos en el mapa marítimo.

1.1. Que es el proyecto Artifact?

ARtifact es un proyecto desarrollado conjuntamente por el Departamento de Arquitectura de Computadoras (EETAC) de la UPC con Renate Rosner rr-consult para explorar conceptos de realidad aumentada o mixta en el mundo marítimo. ARtifact esta desarrollado con Unity3D, SignalK y OpenCPN framework bajo la licencia GPL2. Para el uso final se pretende usar los dispositivos de tipo HMD, pueden ser las gafas de visión aumentada como Microsoft Hololens.

El principal objetivo de ARtifact es integrar realidad aumentada en el mundo marítimo. Las ventajas que puede ofrecer el uso de la tecnología de Realidad Aumentada en un futuro respecto a los sistemas básicos son los siguientes:

- El capitán del barco no tiene que cambiar la vista, mover la cabeza arriba y abajo para buscar la información que desea en el panel de control.
- Pueden mostrar ayudas virtuales o información vital en el campo de visión del usuario.
- Permite una manera de interactuar o coordinar acciones entre varios actores en diferentes lugares.

Este proyecto ha sido como el origen del proyecto llamado Pilotfish.

1.2. Pilotfish

Pilotfish es el primer paso para realizar los objetivos de ARtifact, para aumentar la seguridad durante la navegación marítima. Es un proyecto que no tiene como objetivo la visualización de la información obtenida del barco en ningún dispositivo final concreto, y se pueden utilizarse tanto las gafas de visión aumentada, de realidad virtual, o un PC.

. Con el proyecto Pilotfish, se ha desarrollado el core del proyecto Artifact, en concreto se ha desarrollado un servidor de SignalK que tan solo procesa sentencias NMEA de profundidad y localización y se ha programado un cliente web para PC que permite la suscripción a este servidor de SignalK para poder recibir i visualizar los datos.

Esta primera fase, aunque con sus limitaciones puede proporcionar información útil a los buques cuando naveguen por zonas cuya cartografía de profundidad es poco conocida, para evitar averías y accidentes.

Se ha comprobado el buen funcionamiento del sistema off-line. Para ello, se han recogido y guardado en un fichero los datos (NMEA) de los sensores de profundidad y de localización de una pequeña embarcación que ha realizado

una ruta dentro del Puerto de Port Ginesta. Más tarde, los datos del fichero han alimentado al servidor de SignalK el cual ha generado datos en formato SignalK que han sido enviados a un Cliente que estaba ejecutándose en un PC y que se había suscrito previamente al servidor para recibir los datos. El Cliente cada vez que recibe datos válidos los visualiza de forma gráfica en el PC.

En un futuro, el proyecto está desarrollando un pequeño dron de agua para automatizar el proceso y poder recoger los datos en tiempo real.
La manera de recoger esta información podría ser un dron que navega delante del barco y estudia el fondo, u otra aplicación que podría ser el experimento en el puerto de Ginesta en Castelldefels.

Experimento en Ginesta. Puerto Ginesta está en el mar Mediterráneo a unas 15 mn de Barcelona a sur-oeste. El problema de este puerto consiste en que la profundidad es constantemente variable por las mareas y corrientes que llevan la arena y cambian la cartografía de la profundidad. Consiste en el recorrido del puerto por un dron y estudio de la profundidad en cada punto. Ya que en los últimos años varios barcos se han quedado atascados al entrar al puerto. Teniendo la información que proporciona el dron se podrían actualizar los mapas de profundidad para navegar el barco por la ruta segura en el caso del capitán del barco o mandar un equipo de mantenimiento a este punto en el caso del controlador del puerto.

Para hacer realidad esta prueba en el puerto Ginesta se ha implementado un sensor de profundidad Lowrance Elite 4 DSI Plotter generando NMEA0183 de profundidad y geolocalización, 10” TrekStor® Surftab® Tablet, un modem WiFi Ubiquiti Bullet 5 y un pack de baterías LiPo.

El servidor SignalK se ha instalado en el dron, aunque también podría instalarse en la oficina del capitán o incluso en la nube.

El cliente para obtener los datos del servidor se tiene que registrarse en el servidor y suscribirse a los datos deseados, para poder visualizar la información en una Tablet, PC o incluso en las gafas de visión aumentada.

El puerto Ginesta es muy cómodo para este tipo de prueba ya que tiene varios puntos de WiFi repartidas por las instalaciones. A la hora de testear se han recogido los datos de profundidad por la embarcación y se han podido visualizar desde la oficina del capitán. En el mapa la profundidad es representada por las flechas que cambian de color si la profundidad es menor a la necesaria, para la navegación segura. **(Fig. 1.1)**

Fig. 1.1 Representación de profundidad en el puerto Ginesta

En el caso de peligro puede saltarse un aviso al capitán de que se ha detectado un punto inferior al umbral. Así se evita estar monitorizando la profundidad constantemente.

En el proyecto de visualización de datos en los mapas marítimos se han aumentado las funcionalidades del proyecto Pilotfish.

El proyecto de visualización de datos en mapas marítimos añade al proyecto Pilotfish la posibilidad de visualizar una serie de datos adicionales para funcionalidades y aplicaciones distintas.

Una de ellas es la representación de heading del barco sobre un compás que representa la dirección del barco, y la visualización de la velocidad del barco. Esta funcionalidad puede permitir en un futuro monitorizar varios barcos en el mapa marítimo y estar en alerta sobre en qué dirección y a qué velocidad se acercan los barcos. Esta información podría ser útil en entornos con varias embarcaciones a la vez para vigilar que no se choquen.

Otra información añadida es la velocidad y la dirección del viento. La dirección se representa también sobre la brújula con una flechita. Sus perspectivas por ejemplo para la ayuda a la navegación de un barco de vela, donde si el capitán llevara gafas de visión aumentada podría ver en su campo de visión la información del viento añadida en el tiempo real le podría ayudar a navegar su barco.

El último dato añadido es simplemente la representación de la temperatura del agua.

CAPÍTULO 2. Parsers

2.1. Estándar NMEA

En el proyecto los datos de entrada son recibidos por diferentes sensores de la embarcación:

- Profundidad **(Fig. 2.1)**
- Temperatura **(Fig. 2.1)**
- Viento
- Posición
- Velocidad
- Y muchos más…

Fig. 2.1 Ejemplo de un sensor para barco que mide temperatura y profundidad de la marca HUMMINBIRD.

Los sensores generan sentencias NMEA0183 donde almacenan la información recogida.

NMEA es un protocolo normalizado el cual permite la comunicación entre equipos electrónicos marinos y ordenadores u otros equipos electrónicos marinos. Existen diferentes versiones del estándar NMEA, el usado para la comunicación con *Pilotfish* es el ***NMEA 0183***.

Las sentencias NMEA incluyen la información completa PVT (posición, velocidad y tiempo); cada sentencia enviada es totalmente independiente de las otras. Lo que nos obliga a reunir todas las sentencias NMEA a un solo documento .txt para utilizarlos como datos de entrada al servidor SignalK.

Cada sentencia empieza con el símbolo *'$'* y acaba con un checksum que consiste en *'*'* y dos dígitos hexadecimales para la comprobación del CRC para detectar si ha habido errores en la transmisión de los datos.
Seguido del *'$'* hay un prefijo de dos letras el cual está definiendo el dispositivo que está creando la sentencia NMEA (p.ej. el receptor GPS sería GP).

$**GP**RMC,143058,A,4115.5280,N,00155.2630,E,0.0,245.0,241015,1.3,W*6

También en el proyecto se ha usado el prefijo SD (Sounder, Depth) sobre todo para las sentencias de profundidad.

Las tres letras siguientes que le siguen a la especificación del dispositivo definen el contenido de la sentencia, es decir, la información que se está recibiendo.

La principal causa de la necesidad de la transformación de NMEA a un objeto SignalK es que NMEA es un protocolo propietario y se necesita una licencia para su uso.

2.2. Sentencias utilizadas

Inicialmente en el proyecto Pilotfish se trataban los siguientes tipos de información recibida:

- ***RMC*** - Recommended Minimum Navigation Information

$--**RMC**,hhmmss.ss,A,llll.ll,a,yyyyy.yy,b,k.k,x.x,dddd,m.m,c*hh<CR><LF>

Por ejemplo:

$GPRMC,152053,A,4115.5290,N,00155.2650,E,0.0,268.0,071015,1.3,W*67

Campos que contiene:

- hhmmss: Tiempo UTC (15:20:53)
- Status: A= activo, V = nulo (advertencia del receptor del navegador), P = preciso.
- llll: Latitud
- a: N= norte, S= sud
- yyyyy: Longitud
- b: E= este, W= oeste
- k.k: Velocidad sobre el terreno en nudos
- x.x: Ángulo de la pista en grados

- dddd: Fecha (7 de octubre del 2015)
- m.m: Variación magnética
- c: E= este, W= oeste
- hh: Checksum, siempre empieza por *

- ***DBT*** - Profundidad bajo el transductor

$--**DBT**,d.d,f,m.m,M,f.f,F*hh<CR><LF>

Por ejemplo:

$SDDBT,9.2,f,2.8,M,1.5,F*03

Campos que contiene:

- d.d: Profundidad en pies
- f: pies
- m.m: Profundidad en metros
- M: Metros
- f.f: Profundidad en brazas
- F: Brazas
- hh: Checksum, siempre empieza por *

- ***GLL***- Posición geográfica (latitud y longitud)

$--**GLL**,llll.ll,a,yyyyy.yy,b,hhmmss,F*hh<CR><LF>

Por ejemplo:

$GPGLL,4916.45,N,12311.12,W,225444,A,*1D

Campos que contiene:

- llll: Latitud
- a: N= norte, S= sud
- yyyyy: Longitud
- b: E= este, W= oeste
- hhmmss: Tiempo UTC (22:54:44)
- Status: A= activo, V = nulo (advertencia del receptor del navegador), P = preciso.
- hh: Checksum, siempre empieza por *

Pero para ampliar la funcionalidad del proyecto Pilotfish y poder mostrar más información en el cliente se han añadido los parsers en el servidor SignalK que

tratan la información que proviene de los sensores de temperatura, sensores de heading, velocidad de la embarcación, velocidad y dirección del viento:

- ***MTW*** – Temperatura media del agua

$--**MTW**,x.x,C*hh<CR><LF>

Por ejemplo:

$SDMTW,22.5,C*01

  - x.x: Grados
  - C: Unidad de medida (C = Celsius)
  - hh: Checksum, siempre empieza por *

- ***MWV*** – Velocidad y ángulo del viento

$--**MWV**,x.x,a,x.x,a*hh<CR><LF>

Por ejemplo:

$GPMWV,084.0,T,10.4,K,A*10

Campos que contiene:

  - x.x: Dirección del viento en ángulos, entre 0 y 360
  - a: Referencia, R= Relativa, T = True
  - x.x: Velocidad del viento
  - a: Unidad de la velocidad, K/M/N
  - hh: Checksum, siempre empieza por *

Estas sentencias inicialmente no estaban generadas en el fichero de prueba ya que la embarcación en la que se realizaron las pruebas no estaba este sensor. Las sentencias NMEA0183 fueron generadas manualmente y añadidas al fichero input.txt para poder realizar pruebas en el servidor y mostrar los datos por la pantalla.

- ***VHW*** – Heading del barco y velocidad relativa al agua

$--**VHW**,x.x,T,x.x,M,x.x,N,x.x,K*hh<CR><LF>

Por ejemplo:

$SDVHW,35,T,40,M,0.0,N,45.0,K*71

Campos que contiene:

- x.x: Heading del barco en grados (true)
- T: T = True
- x.x: Heading del barco en grados (magnetic)
- M: M = Magnetic
- x.x: Velocidad relativa al agua (en knots)
- N: N = Knots
- x.x: Velocidad relativa al agua (en km/h)
- K: K = Kilometros
- hh: Checksum, siempre empieza por *

En esta sentencia solo utilizamos el heading true ya que es el que el que nos hace falta. La velocidad relativa al agua no nos interesa ya que utilizamos la velocidad relativa a la tierra.

- ***VTG*** – Curso y velocidad relativa a la tierra

$--**VTG**,x.x,T,x.x,M,x.x,N,x.x,K,m,*hh<CR><LF>

Por ejemplo:

$GPVTG,30.0,T,30.0,M,0.0,N,45.0,K*7F

Campos que contiene:

- x.x: Curso del barco en grados (true)
- T: T = True
- x.x: Curso del barco en grados (magnetic)
- M: M = Magnetic
- x.x: Velocidad relativa a la tierra (en knots)
- N: N = Knots
- x.x: Velocidad relativa a la tierra (en km/h)
- K: K = Kilometros
- hh: Checksum, siempre empieza por *

En esta sentencia NMEA nos interesa la velocidad del barco sobre la tierra.

CAPÍTULO 3. SignalK

SignalK es un formato de datos abierto para el uso marino construido con las tecnologías web como JSON, WebSockets y HTTP. El formato es compatible con el estándar NMEA y las tecnologías de última generación como WiFi, tablets, smartphones e Internet. Es usado tanto para la comunicación entre los instrumentos y sensores de un barco, cómo para el intercambio de dichos datos entre otros barcos, no cómo el protocolo NMEA en el que sólo se usa para el intercambio de datos entre el barco y sus propios instrumentos.

Por lo tanto, SignalK está diseñado para el intercambio de datos entre aplicaciones móviles, web y para conectar los barcos a través de la red.

Los datos SignalK se transmiten en un fichero JSON.

Es un protocolo diseñado para su fácil integración con la Web y aplicaciones móviles y para conectar los barcos modernos al Internet of Things.

Existen varios motivos que han llevado a la aparición del protocolo SignalK como alternativa al protocolo NMEA.

TÉCNICA. Una de ellas es la limitación de la transmisión de datos. El NMEA 0183 está limitado por 4800 baudios (34,800 bps en transmisión rápida) y solo un equipo transmisor. NMEA 2000 funciona mucho más rápido a 250 kbps y soporta hasta 50 dispositivos. Esto estaba muy bien para los años 2000 pero actualmente es insuficiente.

LEGAL. El estándar NMEA teóricamente es “abierto”, pero aun así cualquiera que quiere usarlo no solo tiene que pagar por estándares sino también firmar una licencia que limita como se usa. Lo que no permite el desarrollo de aplicaciones abiertas utilizando estos estándares.

Finalmente y seguramente lo más importante el estándar NMEA y otros protocolos propietarios en la industria han sido desarrollados cuando la instrumentación en un barco mediano era mucho más simple y menos exigente a la capacidad y comunicación con el resto del mundo. Ahora esto parece arcaico que tu barco este aislado del resto del mundo.

Organización. SignalK se divide en tres componentes:

1. El primero es el modelo de datos, que describe una gran jerarquía de puntos de datos organizados en temas y subtemas. Pueden contener datos de su barco, otros barcos en las proximidades, ayudas a la navegación, puntos locales de interés, guías, de hecho cualquier tipo de fuentes de

datos importantes. Tiene una regla general muy importante y que es que cada pieza de datos tiene una ubicación definida, una ruta en la terminología de la señal K. Por ejemplo en el caso de la profundidad es:

```
v.Environment.Depth.BelowTransducer
```

Estos caminos pueden ser considerados como una forma de Uniform Resource Identifier (URI).
Además de los datos dinámicos como la velocidad del viento o las coordenadas GPS, el modelo de datos de la señal K es capaz de almacenar datos estáticos tales como notas de crucero, cartas, clima, avisos, datos de puertos, etc.

2. El segundo componente de SignalK es el modelo de seguridad. Se aplica la separación de permisos como leer, escribir, ejecutar y tres tipos de entidades como usuario, grupo y otro. Se parece en algo a sistema de seguridad de Unix. Pero de hecho solo se aplican los permisos de lectura y escritura, las clasificaciones de usuarios aún no se aplican.
3. El tercer componente es el protocolo SignalK. Esto especifica cómo deben realizarse las interacciones entre varios dispositivos que hablan SignalK. Consta de 5 comandos básicos: list, get, put, subscribe, and unsubscribe.

El modelo de datos suele mostrarse en JSON, y es el que hemos utilizado en nuestro proyecto.

El modelo de datos simplificado puede ser la **(Fig. 3.1)**:


```
{
  "self": "123456789",
  "vessels": {
    "123456789": {
      "name": "motu",
      "mmsi": 123456789,
      "navigation": {
        "headingTrue": {
          "value": 23,
          "source": "self",
          "timestamp": "2014-03-24T00:15:41Z"
        },
        "headingMagnetic": {
          "value": 43,
          "source": "self",
          "timestamp": "2014-03-24T00:15:41Z"
        }
      }
    }
  }
}
```


Fig. 3.1 Modelo de datos

En la **Fig. 3.1** podemos ver:

- Self: Es la clave (MMSI u otra identificación única) del barco, los datos reales se encuentran en el objeto *vessels*.
- Vessels: Objeto contenedor para los barcos, en el que estos son descritos incluyendo el propio barco. Dentro del contenedor tenemos los campos (*name, mmsi, navigation*) los cuales describen el barco y los datos de navegación de éste.
- Navigation: Objeto contenedor de los datos de navegación del barco, este contiene datos como *headingTrue (*dirección actual del barco en grados*)* y *headingMagnetic (*Rumbo magnético actual del barco en grados*)*.
- HeadingTrue y HeadingMagnetic: son campos contenedores con sus propios valores definidos.

El código de SignalK está publicado bajo la Apache License, Version 2.0 y los protocolos e interfaz bajo la licencia de Creative Commons.

Pero al mismo tiempo es un formato nuevo y aun no se ha convertido en estándar, está sujeto a muchas modificaciones y cambios constantemente

Desde que el modelo de datos es consistente, los instrumentos del barco pueden mostrar tus datos a otros y viceversa, Cada instrumento y sensor contiene una copia parcial del modelo, que contiene solo los datos que le interesan. Dado que los instrumentos que soportan SignalK siempre pueden leer los datos, son compatibles entre todos.

Para transmitir los datos entre dispositivos interesados, se utiliza un modelo de árbol jerárquico, con metadatos en la estructura y definición. Es transferido como una sentencia JSON como imagen mostrada anteriormente.

Donde el dispositivo puede preguntar por el heading del barco.

```
vessels.localBoat.navigation.headingTrue
```

O quizás toda la información sobre la navegación:

```
vessels.localBoat.navigation
```

SignalK permite hacer una fusión (merging) de datos entre dispositivos. Permite interacción entre dispositivos, y proporciona una base común para aplicaciones marinas existentes y nuevas.

TRANSPORTE. Como se ha dicho más de una vez SignalK es transmitido como una sentencia JSON. Es muy nativo para el internet y para los navegadores, que permite una transmisión soportada por cualquier medio que tiene acceso a internet. Esto a nuestro proyecto le permite la transferencia de

los datos desde el barco a través de la comunicación WiFi instalada en el puerto a los dispositivos cliente, sean los que sean, el objetivo principal es que en el futuro sea un HMD.

Relación con NMEA. En un futuro podríamos ver los sensores y transductores que utilicen SignalK de forma nativa, pero ahora es necesario una conversión de los datos NMEA0183 y NMEA2000 a SignalK. En el proyecto tenemos en el servidor SignalK un sistema que hace el parser de sentencias NMEA0183 y los guarda en los objetos SignalK correspondientes.

SignalK es un esfuerzo continuo y colaborativo entre una red global de voluntarios y fabricantes de hardware interesados y proveedores de software en la industria naval. Y aún está en desarrollo lo que implica muchos cambios en el modelo. En el proyecto se ha definido una versión concreta del modelo SignalK para hacer el Pilotfish. Debido a cambios frecuentes en el modelo. En un futuro cuando SignalK se convierta en un estándar y se definirá un modelo fijo se aplicara al proyecto Pilotfish aplicando los cambios introducidos en el modelo SignalK definitivo.

CAPÍTULO 4. Servidor SignalK

En el proyecto se ha utilizado el servidor desarrollado inicialmente para el proyecto Pilotfish al cual se han añadido nuevas funcionalidades.
El servidor SignalK (PilotFish SignalK Server – PFSKS) es capaz de transformar las sentencias NMEA a formato JSON siguiendo las especificaciones del formato SignalK.
El servidor almacena todas las sentencias NMEA en un buffer y las convierte en un mensaje SignalK. Todas las sentencias NMEA comprendidas entre dos mensajes de geolocalización (GLL o RMC) son unidas en un objeto (mensaje) SignalK.
El servidor puede leer datos de diferentes fuentes y en diferentes formatos cómo:

- **Fichero de texto:** Dónde cada sentencia NMEA está en una nueva línea.
- **Puerto serie:** El servidor está escaneando continuamente si hay un nuevo puerto serie abierto y comprueba si los datos recibidos son NMEA.
- **Socket TCP:** Cualquier aplicación puede conectarse con el servidor por el puerto 5556 y enviar información a través de un stream de datos NMEA.

Las sentencias NMEA procesadas tal y cómo se ha explicado en el anteriormente (mencionar donde) son:

- **RMC** - Recommended Minimum Navigation Information
- **GLL** - Posición geográfica (latitud y longitud) HDM
- **DBT** - Profundidad bajo el transductor

Al proyecto inicial se han añadido también:

- **MTW** – Temperatura media del agua
- **MWV** – Velocidad y ángulo del viento
- **VHW** – Heading del barco y velocidad relativa al agua
- **VTG** – Curso y velocidad relativa a la tierra

La salida en formato NMEA es a través de:

- Socket TCP por el puerto 55557, la salida es de todas las sentencias comprendidas entre los dos mensajes de geolocalización (GLL o RMC).

La salida del SignalK es a través de:

- Socket TCP por el puerto 55555
- WebSocket por el puerto 3000
- HTTP por el puerto 80

El cliente para recibir los datos deseados tiene que pasar por un proceso de suscripción que consiste en dar saber al servidor que datos quiere recibir.

Esto en el código es representado de la siguiente manera **(Fig. 4.1)**:

```
subscriptionMessage="{\"context\":\"vessels."+Singleton.GetInstance().Mmsi +
"\",\"subscribe\":[{\"path\":\"environment.depth.belowTransducer\"},{\"path\":\"naviga
tion.position\"},{\"path\":\"navigation.headingTrue\"},{\"path\":\"environment.water.te
mperature\"},{\"path\":\"navigation.speedOverGround\"},{\"path\":\"environment.win
d.directionTrue\"},{\"path\":\"environment.wind.speedTrue\"}]}";
```

Fig. 4.1 Suscrición a los datos desde el cliente

En el siguiente esquema general del Pilotfish **(Fig. 4.2)** se puede encontrar en qué lugar se encuentra el servidor.

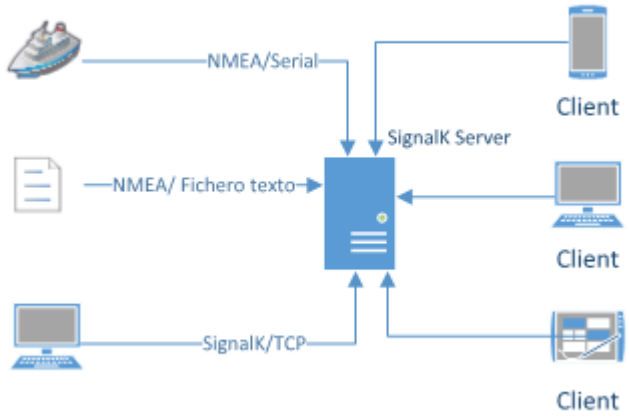

Fig. 4.2 Esquema general del Pilotfish

El servidor puede ejecutarse tanto en la plataforma de ***Windows*** como ***en Linux***, ya que su código se ha generado en el lenguaje de programación C#.

4.1 Parser MTW

Para dar un ejemplo de cómo se hace un parser de datos de NMEA0183 a formato SignalK a continuación se muestra el código de uno de los parsers añadidos al proyecto Pilotfish.

```
// ---------------------------------------------------------------------
// <copyright file="MtwParser.cs" company="UPC">
// TODO: Update copyright text.
// </copyright>
// ---------------------------------------------------------------------

namespace SignalkServer.Parser
{
    using SignalkServer.Data;
    using SignalkServer.Util;
    using Newtonsoft.Json;
```

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;

/// <summary>
/// This class converts a MTW NMEA sentence/s into a single Signalk object.
/// /vessels/<RegExp>/environment/water/temperature SE MIDE EN
KELVINS
/// </summary>
public class MtwParser
{
    /// <summary>
    /// Inner class to store the fields of a MTW sentence.
    /// </summary>
    private class MtwMessage
    {
        public string Degrees { get; set; }
        public string Celcius { get; set; }
    }

    /// <summary>
    /// Merges a MTW NMEA sentence into an existing Signalk object.
    /// </summary>
    /// <param name="sentence">The MTW sentence to merge into the
Signalk object.</param>
    /// <param name="sk">The SignalK object.</param>
    /// <returns>The new Signalk object.</returns>
    public static Signalk Merge(string sentence, Signalk sk)
    {
        Signalk res = null;

        MtwMessage message = new MtwMessage();

        sentence = sentence.Substring(0, sentence.IndexOf("*"));

        string[] info = sentence.Split(',');

        message.Degrees = info[1];
        message.Celcius = info[2];          //Unit of Measurement, Celcius

        if (message.Degrees != "")
        {
            res = MergeMtwMessage(sk, message.Degrees);
        }
        else
        {
            Log.WriteLine("Status: Invalid data.");
        }
```

```
    return res;
}

/// <summary>
/// Writes the MTW heading information in a specific
/// Signalk object.
/// </summary>
/// <param name="sk">The Signalk object.</param>
/// <param name="depth">The value of the depth.</param>
/// <returns>The new Signalk object.</returns>
private static Signalk MergeMtwMessage(Signalk sk, string Degrees)
{
    Vessel v;

    if (sk == null)
    {
        sk = new Signalk();
        sk.Self = "123456789";
    }

    if (sk.Vessels == null)
    {
        sk.Vessels = new Dictionary<string, Vessel>();
    }

    bool res = sk.Vessels.TryGetValue(sk.Self, out v);

    if (!res)
    {
        v = new Vessel();
    }
    ///vessels/<RegExp>/environment/water/temperature
    if (v.Environment == null)
    {
        v.Environment = new Data.Environment();
    }

    if (v.Environment.Water == null)
    {
        v.Environment.Water = new Water();
    }

    if (v.Environment.Water.Temperature == null)
    {
        v.Environment.Water.Temperature = new NumberValue();
    }

    DateTime dt = DateTime.UtcNow;
    string ts = String.Format("{0:s}", dt);
```

```
            ts = ts + "Z";
            v.Environment.Water.Temperature.Value = float.Parse(Degrees,
CultureInfo.InvariantCulture)+273; //Pasar de grados a kelvins K = C + 273
            if (!res)
            {
                sk.Vessels.Add(sk.Self, v);
            }
            return sk;
        }
    }
}
```

CAPÍTULO 5. Cliente SignalK

El cliente SignalK (PilotFish SignalKClient –PFSKC) es una aplicación creada con Unity3D testeada en la plataforma de Windows el cual permite establecer una conexión con el servidor y leer los mensajes SignalK.

La aplicación abre una conexión TCP contra el servidor y lee los mensajes, a los que está suscrito, al principio solo eran posición y profundidad, pero posteriormente se añadieron los datos sobre la temperatura del agua, dirección y velocidad del viento, heading y velocidad del barco. Puede que los mensajes recibidos no contengan información de alguno de los parámetros suscritos ya que como se ha explicado anteriormente, los mensajes SignalK se crean a través de sentencias NMEA comprendidas entre sentencias de geolocalización (RMC o GLL), por lo que cabe la posibilidad que no exista ninguna sentencia NMEA de profundidad (DBT), de temperatura de agua (MTW), velocidad y fuerza del viento (MWV), velocidad relativa a la tierra (VRG) y heading (VHW).

Para representar la profundidad relacionada a geoposición la aplicación cliente crea un mapa con algunos planos y aplica las texturas correspondientes a través de la REST API de OpenStreetMaps.

Una vez el mapa se ha generado y la conexión con el servidor se ha establecido, se procesan los mensajes SignalK dibujando un objeto sobre el mapa con la información recibida (profundidad y geoposición). En el mapa se marca el punto correspondiente con un icono de sonar y la profundidad en dicho punto en texto, el color del icono dependerá de la profundidad en cada punto.

En la **Fig. 5.1** se puede observar la representación en el mapa de los datos procesados de los mensajes recibidos.

Fig. 5.1 Vista del mapa, representación de profundidad y geoposición.

Para representar los nuevos datos introducidos al proyecto Pilotfish se utiliza una brújula **(Fig. 5.2)** sobre la cual podemos visualizar fácilmente la información sobre el heading del barco (imagen del barco) y una flecha azul que indica la dirección del viento.

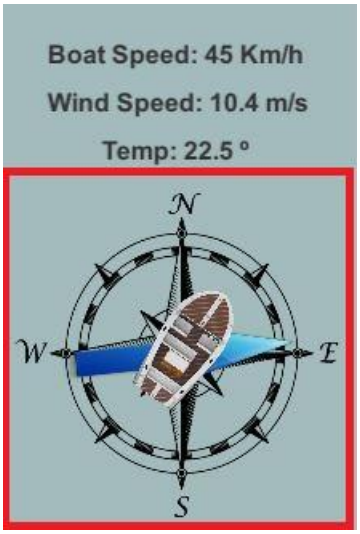

Fig. 5.2 Compas con el heading del barco y la dirección del viento.

Por encima del compás podemos observar la información sobre a qué velocidad se mueve la embarcación **(Fig. 5.3)**.

- Velocidad del barco en Km/h (Boat speed)
- Velocidad del viento en m/s (Wind speed)
- Temperatura del agua en Cº (Temp)

Fig. 5.3 Información sobre la velocidad del barco, viento y temperatura del agua.

La información se va actualizando cada vez que se recibe nueva sentencia NMEA con nuevos datos, si en el periodo comprendido entre sentencias de geolocalización (RMC o GLL) no se ha encontrado la sentencia NMEA para actualizar los datos estos datos se mantienen de la sentencia anterior hasta

recibir una más actualizada. Esto es debido a que diferentes sensores del barco generan las sentencias con diferente periodicidad.

5.1 Código clave del cliente.

En el cliente la parte más importante del código es la que hace la suscripción a los datos (Fig. 4.1) y el código que hace el parser de la sentencia JSON extrayendo los datos de profundidad, temperatura, viento, velocidad y heading del barco:

```
private void ReadUpdateMessage(string info)
{
    using (StringReader stringReader = new StringReader(info))
    using (JsonTextReader jsonReader = new JsonTextReader(stringReader))
    {
        float depth = 0.0F;
        float latitude = 0.0F;
        float longitude = 0.0F;
        string timestamp = "";
        float temperature = 0.0F;
        float boatspeed = 0.0F;
        float windspeed = 0.0F;
        float heading = 0.0F;
        float windangle = 0.0F;

        bool nextDepth = false;
        bool nextLatitude = false;
        bool nextLongitude = false;
        bool nextTimestamp = false;
        bool nextTemperature = false;
        bool nextBoatspeed = false;
        bool nextWindspeed = false;
        bool nextHeading = false;
        bool nextWindangle = false;

        while (jsonReader.Read())
        {
            if (nextDepth)
            {
                if (jsonReader.TokenType.ToString() == "Float")
                {
                    depth = float.Parse(jsonReader.Value.ToString());
                    nextDepth = false;
                }
            }
            if (nextLongitude)
            {
                longitude = float.Parse(jsonReader.Value.ToString());
```

```
            nextLongitude = false;
        }
        if (nextLatitude)
        {
            latitude = float.Parse(jsonReader.Value.ToString());
            nextLatitude = false;
        }
        if (nextTimestamp)
        {
            timestamp = jsonReader.Value.ToString();
            nextTimestamp = false;
        }
        if (nextBoatspeed)
        {
            if (jsonReader.TokenType.ToString() == "Float")
            {
                boatspeed = float.Parse(jsonReader.Value.ToString());
                nextBoatspeed = false;
            }
        }
        if (nextTemperature)
        {
            if (jsonReader.TokenType.ToString() == "Float")
            {
                temperature = float.Parse(jsonReader.Value.ToString());
                nextTemperature = false;
            }
        }
        if (nextWindspeed)
        {
            if (jsonReader.TokenType.ToString() == "Float")
            {
                windspeed = float.Parse(jsonReader.Value.ToString());
                nextWindspeed = false;
            }
        }
        if (nextHeading)
        {
            if (jsonReader.TokenType.ToString() == "Float")
            {
                heading = float.Parse(jsonReader.Value.ToString());
                nextHeading = false;
            }
        }
        if (nextWindangle)
        {
            if (jsonReader.TokenType.ToString() == "Float")
            {
                windangle = float.Parse(jsonReader.Value.ToString());
                nextWindangle = false;
```

```
                }
            }

            if (jsonReader.Value!= null && jsonReader.Value.ToString() ==
"environment.depth.belowTransducer")
            {
                nextDepth = true;
            }
            if (jsonReader.Value != null && jsonReader.Value.ToString() ==
"environment.water.temperature")
            {
                nextTemperature = true;
            }
            if (jsonReader.Value != null && jsonReader.Value.ToString() ==
"navigation.speedOverGround")
            {
                nextBoatspeed = true;
            }
            if (jsonReader.Value != null && jsonReader.Value.ToString() ==
"environment.wind.speedTrue")
            {
                nextWindspeed = true;
            }
            if (jsonReader.Value != null && jsonReader.Value.ToString() ==
"navigation.headingTrue")
            {
                nextHeading = true;
            }
            if (jsonReader.Value != null && jsonReader.Value.ToString() ==
"environment.wind.directionTrue")
            {
                nextWindangle = true;
            }
            if (jsonReader.TokenType.ToString() == "PropertyName" &&
jsonReader.Value.ToString() == "longitude")
            {
                nextLongitude = true;
            }
            if (jsonReader.TokenType.ToString() == "PropertyName" &&
jsonReader.Value.ToString() == "latitude")
            {
                nextLatitude = true;
            }
            if (jsonReader.TokenType.ToString() == "PropertyName" &&
jsonReader.Value.ToString() == "timestamp")
            {
                nextTimestamp = true;
            }
        }
        Signalk sk = new Signalk();
```

```
            sk.Self = Singleton.GetInstance().Mmsi;
            Vessel v = new Vessel();
            v.Environment = new Environment();
            v.Environment.Depth = new Depth();
            v.Environment.Water = new Water();
            v.Environment.Wind = new Wind();
            v.Environment.Depth.BelowTransducer = new NumberValue();
            v.Environment.Water.Temperature = new NumberValue();
            v.Environment.Wind.SpeedTrue = new NumberValue();
            v.Environment.Wind.DirectionTrue = new NumberValue();
            v.Environment.Depth.BelowTransducer.Value = depth;
            v.Environment.Water.Temperature.Value = temperature;
            v.Environment.Wind.SpeedTrue.Value = windspeed;
            v.Environment.Wind.DirectionTrue.Value = windangle;
            v.Navigation = new Navigation();
            v.Navigation.SpeedOverGround = new NumberValue();
            v.Navigation.HeadingTrue = new NumberValue();
            v.Navigation.HeadingTrue.Value = heading;
            v.Navigation.SpeedOverGround.Value = boatspeed;
            v.Navigation.Position = new Position();
            v.Navigation.Position.Latitude = latitude;
            v.Navigation.Position.Longitude = longitude;
            v.Navigation.Position.Timestamp = timestamp;
            sk.Vessels = new Dictionary<string, Vessel>();
            sk.Vessels.Add(sk.Self, v);
            Singleton.GetInstance().AddSignalk(sk);
            double tem = v.Environment.Water.Temperature.Value - 273;
            string tempString = tem.ToString();
            if (tempString.Length > 5)
            {
                tempString = tempString.Substring(0, 5);
            }
            Singleton.GetInstance().Temperature = ("Temp: " + tempString + " º");

            if(v.Navigation.SpeedOverGround.Value != 0)
            {
                Singleton.GetInstance().BoatSpeed = ("Boat Speed: " +
v.Navigation.SpeedOverGround.Value + " Km/h");
            }

            if (v.Environment.Wind.SpeedTrue.Value != 0)
            {
                Singleton.GetInstance().WindSpeed = ("Wind Speed: " +
v.Environment.Wind.SpeedTrue.Value + " m/s");
            }

            if (windangle != 0)
            {
                Singleton.GetInstance().WindAngle = windangle + "";
            }
```

```
            if (heading != 0)
            {
                double headingdegrees = heading * 180 / Math.PI;
                Singleton.GetInstance().Heading = headingdegrees + "";
            }
        }
    }
```

5.2 Interfaz de usuario.

El cliente tiene varios ajustes en la parte de UI **(Fig. 5.4)**.

En un principio los elementos añadidos por el proyecto de visualización de datos no estaban como en la **Fig. 5.4** pero en la imagen **Fig. 5.5** y **Fig. 5.6** ya están y se pueden apreciar los cambios introducidos por el proyecto.

El cliente ofrece al usuario diferentes opciones para interactuar con el mapa, de esta manera el usuario a través del UI puede por ejemplo hacer zoom sobre el mapa, guardar una imagen de éste o saber si se está recibiendo información del servidor.
A continuación, explicamos cada una de las opciones que se ofrecen al usuario para la interacción con la aplicación cliente:

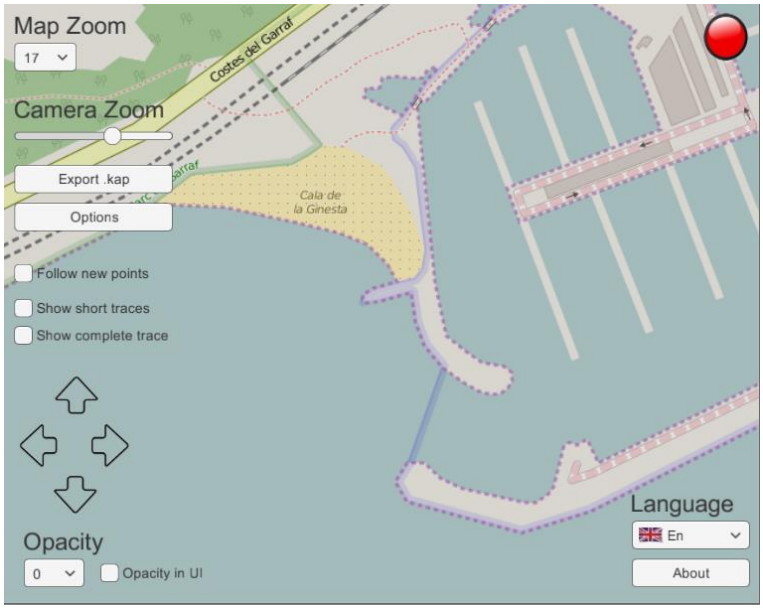

Fig. 5.4 Mapa con los controles de UI

ZOOM

Menú desplegable que permite seleccionar el zoom deseado. Cuando el valor del zoom se cambia, realmente lo que se hace es descargar un nuevo tile de

OpenStreetMap, el mapa está formado por diferentes tiles descargados de los servidores de OpenStreetMap.

Camera Zoom

Este zoom no cambia el espacio representado por un tile, realmente lo que se está haciendo es ampliar/disminuir el pixel representado en el mapa.

Export .Kap

Esta opción permite tener una captura de pantalla sin los elementos del UI ni el fondo del mapa, la captura de pantalla se guarda en un fichero PNG, a su vez llama a la aplicación imgkap para poder crear el fichero KAP del fichero PNG creado con las coordenadas de la vista que se tienen en la aplicación.

Options

Muestra una nueva ventana para poder seleccionar el destino donde serán guardados los ficheros PNG y imgkap. En la **Fig. 5.4** podemos observar el formato que tiene dicha opción.

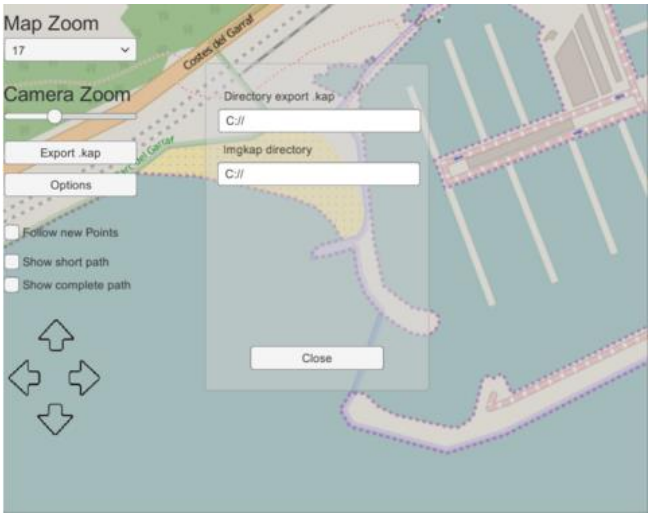

Fig. 5.4 Menú de opciones

Botones de activación

- Follow new points: Si es activado, cada vez que la aplicación dibuja un nuevo objeto la cámara se centrara en el nuevo objeto automáticamente.

- Show short path: Si es activado, todos los puntos serán unidos con líneas mostrando el recorrido del barco. El color de las líneas dependerá de la profundidad del punto. En la **Fig. 5.5** se muestra

cual sería el resultado final. Tiene exclusividad mutua con Show complete path, solo puede haber uno activado.

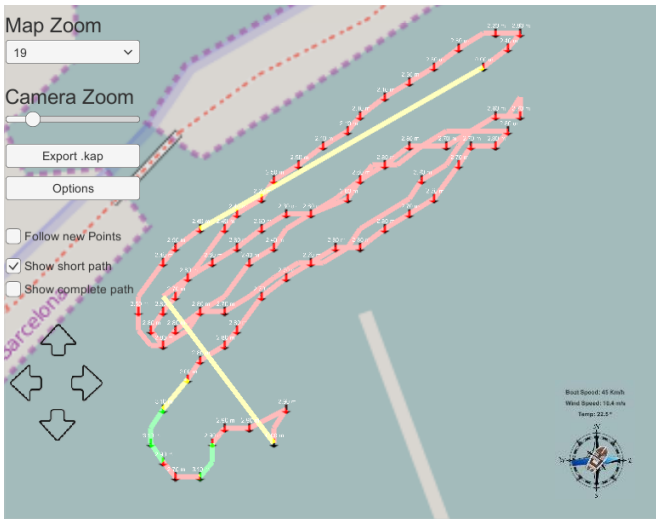

Fig. 5.5 Short trace

- Show complete path: Si es activado, todos los puntos serán unidos con una línea verde, de esta manera se ve todo el recorrido del barco. En la **Fig. 5.6** se muestra cómo quedaría la traza completa del barco. Tiene exclusividad mutua con Show short path, solo puede haber uno activado.

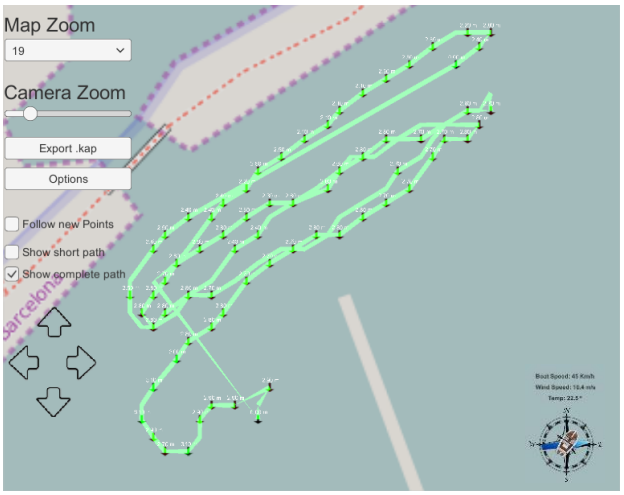

Fig. 5.6 Complete trace

Flechas de movimiento

Permite poder desplazarse a través del mapa, en el caso de que moviéndose a través del mapa se salga de los límites establecidos por los tiles cargados, se descargará el siguiente tile consecutivo a la zona del mapa.

Opacidad

Selecciona la luminosidad de la pantalla en diferentes niveles.

- Opacity in UI: Si está activado, a los controles del UI también se les aplicará la opacidad seleccionada por el usuario.

Estado de conexión

Situado en la parte superior derecha del UI, el led consta de los diferentes estados:

- Rojo: Indica que el cliente no está recibiendo información del servidor, es decir, está desconectado.
- Verde: Indica que el cliente recibe información del servidor.

CAPÍTULO 6. Conclusiones

El proyecto ARtifact tiene buenas perspectivas en el ámbito de realidad aumentada aplicada a la navegación marítima. Se ha podido dar un paso más ampliando el proyecto Pilotfish agregando nuevas funcionalidades. Que podrán en un futuro ampliar las aplicaciones de visualización de datos de navegación en realidad aumentada. Además de esto se ha visto las futuras perspectivas del formato SignalK sobre el actual NMEA que en un futuro podrá convertirse en un estándar y aparecer en muchos dispositivos y empezar a ser un protocolo de uso global ya que su uso libre abre muchas puertas en el mercado.

Bibliografía

[1] http://catb.org/gpsd/NMEA.html

[2] http://signalk.org/

[3] https://unity3d.com/

[4] https://www.visualstudio.com/

[5] https://en.wikipedia.org/wiki/NMEA_0183

[6] https://www.visualstudio.com/

[7] Pilotfish.doc Lidia Bern, David Nuevo, Dolors Royo. 2016 EETAC "Descripción del proyecto Pilotfish"

[8] ARtifact.doc Lidia Bern, David Nuevo, Dolors Royo. 2016 EETAC "Descripción del proyecto ARtifact"